

```

;*****
;   Virtual Peripheral UART
;
;
;   Length: 67 bytes (total)
;   Authors: Chip Gracey, President, Parallax Inc.
;            modified by Craig Webb, Consultant to Scenix Semiconductor, Inc.
;   Written: 97/03/10 to 98/7/09
;
;*****
;***** Assembler directives
;
; uses: SX28AC, 2 pages of program memory, 8 banks of RAM, high speed osc.
;       operating in turbo mode, with 8-level stack & extended option reg.
;
;       DEVICE pins28,pages2,banks8,oschs
;       DEVICE turbo,stackx,optionx
;       ID      'UART'                ;program ID label
;       RESET   reset_entry           ;set reset/boot address
;
;***** Program Variables *****
;
; Port Assignment: Bit variables
;
rx_pin      EQU      ra.2             ;UART receive input
tx_pin      EQU      ra.3             ;UART transmit output
;
;***** Register definitions (bank 0)
;
;       org      8                    ;start of program registers
main        =      $                  ;main bank
;
temp        ds       1                ;temporary storage
byte        ds       1                ;temporary UART/I2C shift reg.
flags       DS       1                ;program flags register
number_low  ds       1                ;low byte of rec'd value
number_high ds       1                ;high byte of rec'd value
hex         ds       1                ;value of rec'd hex number
string      ds       1                ;indirect ptr to output string
;
rx_flag     EQU      flags.5          ;signals when byte is received
;
;
;       org      10h                  ;bank3 variables
serial     =      $                  ;UART bank
;
tx_high     ds       1                ;hi byte to transmit
tx_low      ds       1                ;low byte to transmit
tx_count    ds       1                ;number of bits sent
tx_divide   ds       1                ;xmit timing (/16) counter
rx_count    ds       1                ;number of bits received
rx_divide   ds       1                ;receive timing counter
rx_byte     ds       1                ;buffer for incoming byte
;
; The following three values determine the UART baud rate.
; The value of baud_bit and int_period affect the baud rate as follows:
; Baud rate = 50MHz/(2^baud_bit * int_period * RTCC_prescaler)

```

```

;      Note:  1 =< baud_bit =< 7
;              *int_period must <256 and longer than the length of the slowest
;                  possible interrupt sequence in instruction cycles.
;                  Changing the value of int_period will affect the
;                  rest of the virtual peripherals due to timing issues.
; The start delay value must be set equal to (2^baud_bit)*1.5 + 1
;
; *** 2400 baud (for slower baud rates, increase the RTCC prescaler)
;baud_bit      =      7                      ;for 2400 baud
;start_delay    =      128+64+1              ; " " "
;int_period     =      163                  ; " " "
;
; *** 9600 baud
;baud_bit      =      5                      ;for 9600 baud
;start_delay    =      16+8+1               ; " " "
;int_period     =      163                  ; " " "
;
; *** 19200 baud
;baud_bit      =      4                      ;for 19200 baud
;start_delay    =      16+8+1               ; " " "
;int_period     =      163                  ; " " "
;
; *** 38400 baud
;baud_bit      =      3                      ;for 38400 baud
;start_delay    =      8+4+1                ; " " "
;int_period     =      163                  ; " " "
;
; *** 57600 baud
;baud_bit      =      2                      ;for 57600 baud
;start_delay    =      4+2+1                ; " " "
;int_period     =      217                  ; " " "
;
; *** 115.2k baud
;baud_bit      =      1                      ;for 115.2K baud
;start_delay    =      2+1+1                ; " " "
;int_period     =      217                  ; " " "
;
; *** 230.4k baud (for faster rates, reduce int_period - see above*)
;baud_bit      =      0                      ;for 230.4K baud
;start_delay    =      1+0+0                ; " " "
;int_period     =      217                  ; " " "
;
;***** INTERRUPT CODE *****
;
; Note: The interrupt code must always originate at 0h.
; Care should be taken to see that any very timing sensitive routines
; (such as adcs, etc.) are placed before other peripherals or code
; which may have varying execution rates (like the UART, for example).
;
interrupt      ORG      0                      ;interrupt starts at 0h
;
;
;**** Virtual Peripheral: Universal Asynchronous Receiver Transmitter (UART)
;
; This routine sends and receives RS232C serial data, and is currently
; configured (though modifications can be made) for the popular
; "No parity-checking, 8 data bit, 1 stop bit" (N,8,1) data format.
; RECEIVING: The rx_flag is set high whenever a valid byte of data has been
; received and it the calling routine's responsibility to reset this flag
; once the incoming data has been collected.

```

```
; TRANSMITTING: The transmit routine requires the data to be inverted
; and loaded (tx_high+tx_low) register pair (with the inverted 8 data bits
; stored in tx_high and tx_low bit 7 set high to act as a start bit). Then
; the number of bits ready for transmission (10=1 start + 8 data + 1 stop)
; must be loaded into the tx_count register. As soon as this latter is done,
; the transmit routine immediately begins sending the data.
; This routine has a varying execution rate and therefore should always be
; placed after any timing-critical virtual peripherals such as timers,
; adcs, pwms, etc.
; Note: The transmit and receive routines are independent and either may be
; removed, if not needed, to reduce execution time and memory usage,
; as long as the initial "BANK serial" (common) instruction is kept.
```

```
;
; Input variable(s) : tx_low (only high bit used), tx_high, tx_count
; Output variable(s) : rx_flag, rx_byte
; Variable(s) affected : tx_divide, rx_divide, rx_count
; Flag(s) affected : rx_flag
; Size : Transmit - 15 bytes + 1 byte shared with receive code
;         Receive - 20 bytes + 1 byte shared with transmit code
; Timing (turbo) :
;         Transmit - (a) [not sending] 9 cycles
;                   (b) [sending] 19 cycles
;                   + 1 cycle shared with RX code ("bank" instr.)
;         Receive - (a) [not receiving] 9 cycles
;                   (b) [start receiving] 16 cycles
;                   (c) [receiving, awaiting bit] 13 cycles
;                   (d) [receiving, bit ready] 17 cycles
;
```

```
bank serial ;switch to serial register bank

:transmit      clr    tx_divide.baud_bit    ;clear xmit timing count flag
               inc    tx_divide             ;only execute the transmit routine
               STZ                    ;set zero flag for test
               SNB    tx_divide.baud_bit    ; every 2^baud_bit interrupt
               test   tx_count             ;are we sending?
               JZ     :receive             ;if not, go to :receive
               clc                    ;yes, ready stop bit
               rr     tx_high              ; and shift to next bit
               rr     tx_low
               dec    tx_count             ;decrement bit counter
               movb   tx_pin,tx_low.6      ;output next bit

;
:receive      movb   c,rx_pin              ;get current rx bit
               test   rx_count             ;currently receiving byte?
               jnz    :rxbit              ;if so, jump ahead
               mov     w,#9                ;in case start, ready 9 bits
               sc     ;skip ahead if not start bit
               mov     rx_count,w          ;it is, so renew bit count
               mov     rx_divide,#start_delay ;ready 1.5 bit periods
:rxbit        djnz   rx_divide,:rxdone     ;middle of next bit?
               setb   rx_divide.baud_bit   ;yes, ready 1 bit period
               dec    rx_count             ;last bit?
               sz     ;if not
               rr     rx_byte              ; then save bit
               snz    ;if so
               setb   rx_flag              ; then set flag

:rxdone
;
               mov     w,#-int_period      ;interrupt every 'int_period' clock
```

01000100

00110100

① 10011010

```

:end_int          retiw          ;exit interrupt
;
;***** End of interrupt sequence
;
;***** PROGRAM DATA *****
;
; String data for user interface (must be in lower half of memory page)
;
_hello           dw          13,10,13,10,'SX 2400-230.4K UART Virtual Peripheral Demo'
_cr              dw          13,10,0
_prompt         dw          13,10,'>',0
_error          dw          'Error!',13,10,0
_hex            dw          '0123456789ABCDEF'
_space          dw          ' ',0
_sample         dw          13,10,'Sample=',0
_view           dw          13,10,'Bytes stored:',0
;
;***** SUBROUTINES *****
;
; Subroutine - Get byte via serial port
;
get_byte         jnb         rx_flag,$          ;wait till byte is received
                clrb        rx_flag          ;reset the receive flag
                mov         byte,rx_byte      ;store byte (copy using W)
                ; & fall through to echo char back
;
; Subroutine - Send byte via serial port
;
send_byte        bank       serial
;wait           test        tx_count          ;wait for not busy
                jnz         :wait            ;
                not         w                ;ready bits (inverse logic)
                mov         tx_high,w         ; store data byte
                setb        tx_low.7         ; set up start bit
                mov         tx_count,#10      ;1 start + 8 data + 1 stop bit
                RETP          ;leave and fix page bits
;
; Subroutine - Send string pointed to by address in W register
;
send_string      mov         string,w          ;store string address
:loop           mov         w,string          ;read next string character
                mov         m,#0             ; with indirect addressing
                iread        ; using the mode register
                mov         m,$F             ;reset the mode register
                test        w                ;are we at the last char?
                snz          ;if not=0, skip ahead
                RETP          ;yes, leave & fix page bits
                call        send_byte        ;not 0, so send character
                inc         string           ;point to next character
                jmp         :loop            ;loop until done
;
;***** MAIN PROGRAM CODE *****
;
;                ORG         100h
;

```

```

; Program execution begins here on power-up or after a reset
;
reset_entry
    mov     ra, #%1011           ;initialize port RA
    mov     !ra, #%0100         ;Set RA in/out directions
    CLR     FSR                 ;reset all ram starting at 08h
:zero_ram  SB     FSR.4          ;are we on low half of bank?
    SETB    FSR.3              ;If so, don't touch regs 0-7
    CLR     IND                 ;clear using indirect addressing
    IJNZ    FSR, :zero_ram      ;repeat until done

    mov     !option, #%10011111 ;enable rtcc interrupt
;
; Main Program Loop
;
    mov     w, #_hello          ;send hello string
    page    send_string
    call    send_string

    mov     w, #_prompt         ;send prompt
    page    send_string
    call    send_string

:loop      call    get_byte      ;get byte via UART

;
; <program code goes here>
;
    JMP     :loop              ;back to main loop
;
;*****
END                               ;End of program code

```